FAT-Pointer based range addresses

AKILAN SELVACOUMAR, Heriot Watt University, UK

The increasing disparity between application workloads and the capacity of Translation Lookaside Buffers (TLB) has prompted researchers to explore innovative solutions to mitigate this gap. One such approach involves leveraging physically contiguous memory to optimize TLB utilization. Concurrently, advancements in hardware-level system security, exemplified by the Capability Hardware Enhanced RISC Instructions (CHERI) architecture, offer additional opportunities for improving memory management and security.

CHERI introduces capability-based addressing, a novel approach that enhances system security by associating capabilities with memory pointers. These capabilities restrict access to memory regions, thereby fortifying the system against various security threats. Importantly, the mechanisms implemented in CHERI for enforcing memory protection can also serve as accelerators for standard user-space memory allocators. By leveraging capability-based addressing, memory allocators can efficiently manage memory resources while ensuring robust security measures are in place.

ACM Reference Format:
Akilan Selvacoumar. 2018. FAT-Pointer based range addresses. *J. ACM* 37, 4, Article 111 (August 2018), 3 pages.
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

In the dynamic landscape of computing, the pursuit of optimal performance is a constant endeavor, especially as applications evolve to handle increasingly complex workloads. One critical aspect influencing performance is memory management, where efficient utilization of resources is paramount. Translation Lookaside Buffers (TLBs) play a pivotal role in this regard, expediting memory access by storing recently accessed memory translations. However, as applications grow in size and complexity, the capacity of TLBs often struggles to keep pace, leading to performance bottlenecks. To address this challenge, researchers have turned to innovative solutions, one of which involves harnessing the benefits of huge pages. Huge pages, also known as large pages, allow for the allocation of memory in significantly larger chunks compared to traditional small pages. By reducing the number of TLB entries needed to access a given amount of memory, huge pages offer a potential avenue for optimizing TLB utilization and thereby enhancing overall system performance.

Simultaneously, advancements in hardware-level security, such as the Capability Hardware Enhanced RISC Instructions (CHERI) architecture, present additional opportunities for performance enhancement. CHERI's capability-based addressing approach not only strengthens system security by tightly controlling memory access but also provides avenues for accelerating memory management operations.

In this context, the integration of huge pages into memory management strategies alongside capability-based addressing in architectures like CHERI offers a compelling synergy. By optimizing TLB utilization through the utilization of huge pages and leveraging the security features of

Author's address: Akilan Selvacoumar, as251@hw.ac.uk, Heriot Watt University, UK.

0004-5411/2018/8-ART111 $15.00
https://doi.org/XXXXXXX.XXXXXXX

capability-based addressing, significant performance improvements can be realized. This approach not only enhances system security but also accelerates memory access.

1.1 Background

Efficient memory management, particularly in the context of Translation Lookaside Buffer (TLB) optimization, has been a focal point of research and development within computer architecture. Various techniques have been proposed to mitigate TLB-related bottlenecks and improve overall system performance.

1.1.1 Huge Pages: This is used to map a very large region of memory to a single entry. This small/large region of memory is physically contiguous. Most implementations of huge pages are size aligned, For example for the x86 architecture the huge pages size are 4KB, 2MB and 1GB pages.

1.1.2 Segment: A segment can be viewed as mapping between contiguous virtual memory and contiguous physical memory. The property of a segment allows it to be larger than a page. Direct Segment <paper reference> allows the user to set a single segment for an application. Two registers are added to mark the start and end of the segment. Any virtual address within this region can be translated by adding the fixed offset between the virtual and physical address.

1.1.3 RMM:. RMM introduces the concept of adding an additional range table. For large allocations RMM eagerly allocates contiguous physical pages. The following allocations creates large memory ranges that are both virtually and physically contiguous. RMM builds on the concept of Direct segment <paper reference> by adding offset to translate a virtual address to physical address. RMM compares address with range boundaries to decide which range it belongs to. RMM queries the range table ofter an L1 TLB miss.

1.1.4 FlexPointer: FlexPointer is based on the RMM<paper reference RMM> paper. FlexPointer does eagerly allocate pages which are physically contiguous and stores the ID to translate a virtual address to physical address on the remaining unused bits on the 64 bit virtual address. The paper contribution mentions shifting the TLB lookup to an earlier stage to improve latency of accessing the TLB entries. FlexPointer immediate queries the range TLB for translations rather than the RMM paper which waits for the L1 TLB miss.

2 FAT-POINTER BASED RANGE ADDRESSES

This experiment exploits the FAT pointer to enable faster

2.1 Overview

FAT-Pointers, combined with the capabilities of the CHERI (Capability Hardware Enhanced RISC Instructions) architecture, introduce robust memory safety and security features by incorporating additional metadata with memory pointers. This enhanced architecture utilizes concepts such as FlexPointer, Range Memory Mapping (RMM), and Direct segment to manage memory effectively.

Range addresses play a pivotal role within this framework, defining memory regions bounded by a starting address (BASE) and an ending address (LIMIT). These range addresses are encoded within FAT-pointers, allowing for precise control over memory access permissions.

The functionality of ranges encompasses several key aspects:

- **Creation of Physically Contiguous Memory Ranges**: By defining memory regions that are physically contiguous, systems can achieve optimal memory access patterns, enhancing performance and efficiency.

- **Encoding Ranges as Bounds to the Pointer**: Integrating range bounds directly into FAT-pointers enables the architecture to enforce memory access restrictions at the pointer level thus allowing tracking of memory ranges on a pointer level.
- **Instrumenting Block-Based Allocators with Physically Contiguous Memory**: The integration of range-based memory concepts into memory allocation systems, such as block-based allocators, facilitates the efficient management and utilization of physically contiguous memory blocks, mitigating issues related to memory fragmentation.

2.2 Range creation

Ranges of memory are created based on bounds encoded to the FAT-Pointer. The Chuck between the following bounds is physically contiguous. This is done via the FreeBSD kernel Contig memory allocator.

2.3 Fragmentation

The problem with standard allocators which are physically contiguous is that they are mostly on the

2.4 Allocation with huge pages

3 EVALUATION

Metric name	type of graph	tool used	x axis	y axis
DTLB L1 read	line graph	Pmcstat	Time	DTLB L1 reads (each second)
DTLB L2 read	line graph	Pmcstat	Time	DTLB L2 reads (each second)
DTLB walk	line graph	Pmcstat	Time	DTLB Walks (each second)
L1 cache miss	line graph	Pmcstat	Time	L1 cache miss (each second)
Wall clock run time	bar graph	time	Benchmarks	Time
Resident memory usage	line graph	ps with rss	Time	Memory in MB
Speed ups	bar graph	time	Benchmarks	Time

Benchmark name	Benchmark metrics extracted
Kmeans (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses <
Histogram (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses
Matrix multiply (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses
Matrix multiply (Hans)	
Word count (Coz)	
Sqlite (Coz)	

4 LIMITATIONS

5 RELATED WORK

6 SUMMARY

ACKNOWLEDGMENTS

To Robert, for the bagels and explaining CMYK and color spaces.

REFERENCES